EXHIBIT 44

(Part 1 of 2)

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

ARISTA NETWORKS, INC.
Petitioner

v.

CISCO SYSTEMS, INC.
Patent Owner

Case No. IPR2016-00119
Patent No. 7,047,526

PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 7,047,526

Mail Stop PATENT BOARD Patent Trial and Appeal Board U.S. Patent & Trademark Office P.O. Box 1450 Alexandria, VA 22313-1450	Submitted by: Keker & Van Nest LLP 633 Battery Street San Francisco, CA 94111 (415) 391-5400 (tel) (415) 397-7188 (fax) Counsel for Petitioner Arista Networks, Inc. November 4, 2015

TABLE OF CONTENTS

TABLE OF AUTHORITIES

Page(s)

Federal Cases

Federal Statutes

Federal Regulations

EXHIBIT LIST

Exh. No.	Description
1001	U.S. Patent No. 7,047,526 to Wheeler et al., issued May 16, 2006 ("'526 Patent").
1002	U.S. Patent No. 6,523,172 to Martinez-Guerra et al., issued Feb. 18, 2003 ("Martinez-Guerra").
1003	Noam Chomsky & George A. Miller, *Finite State Language*, 1 Information and Control 91-112 (1958).
1004	Noam Chomsky, *On Certain Formal Properties of Grammars*, 2 Information and Control 137-167 (1959).
1005	Alfred V. Aho, et. al., Compilers: Principles, Techniques, And Tools (1986).
1006	Gerry Kane, MIPS RISC Architecture (1987).
1007	OpenVMS User's Manual, version 7.1 (Nov. 1996), http://www.mi.infn.it/~calcolo/OpenVMS/ssb71/6489/6489p.htm.
1008	Honeywell Bull, Multics: Commands and Active Functions (1985).
1009	Brian W. Kernighan & Rob Pike, The UNIX Programming Environment (1984).
1010	Cisco's Preliminary Claim Construction Disclosure, *Cisco Systems, Inc. v. Arista Networks, Inc.*, No. 5:14-cv-05344-BLF (N.D. Cal. Aug. 24, 2015).
1011	Robert Sedgwick, Algorithms in C (3rd ed. 1998).
1012	Dennis M. Ritchie & Ken Thompson, The UNIX Time-Sharing System, 17 Communications of the ACM 365 (Jul. 1974).
1013	U.S. Patent No. 6,134,709 to Pratt, issued Oct. 17, 2000.
1014	Declaration of Douglas W. Clark, Ph.D.

I. INTRODUCTION

Arista Networks Inc. ("Petitioner") petitions for *inter partes* review of claims 1-26 ("the challenged claims") of United States Patent No. 7,047,526 to Wheeler *et al.*, entitled "Generic Command Interface for Multiple Executable Routines" ("'526 patent") (Ex. 1001).

The '526 patent is directed to a system and method for parsing "generic commands" received from a user and translating them into "prescribed commands" that are issued to a selected "management program." (*Id.*, Abstract.) The specification explains that "system administrators may attempt to utilize multiple tools within a software system . . . for improved system performance." (*Id.* at 1:28-31.) A problem with using many tools, however, is that it "requires the users to remember the names and syntaxes of numerous commands for the respective . . . programs and . . . tools." (*Id.* at 1:32-34.)

The '526 patent purports to solve this problem by providing a system and method for integrating multiple tools and programs "without the necessity of learning the respective command formats and syntax." (*Id.* at 1:43-44.) It claims a system that includes a "command parse tree" configured to parse a "generic command" received from a user. (*Id.* at 1:50-54.) According to the specification, a "generic command" is "an abstraction of the tool-specific command formats and

syntax, enabling a user to issue command[s] based on the relative functions, as opposed to the specific syntax for a corresponding tool." (*Id.* at 3:32-35.)

"The command parse tree includes multiple elements, each specifying at least one corresponding generic command component and a corresponding at least one command action value." (*Id.* at 1:51-54.) The parser identifies the element that's the "best match" for the generic command, then "issues a prescribed command for a selected one of the management programs according to [its] corresponding command format based on the selected command action value." (*Id.* at 1:54-58.) According to the patent, this allows a user to "control multiple management programs having respective command formats, by using a set of generic commands that are independent from the command formats, eliminating the necessity that the user needs to learn the detailed command formats and syntax." (*Id.* at 1:58-63.)

The '526 patent claimed nothing new when it was applied for in June 2000. The idea of translating commands from a high-level format into a tool-specific format for a particular component is fundamental to computer science, and had been known for decades. To cite just one of many examples, Digital Equipment Corporation's "Digital Command Language," or "DCL," was used since the 1980s to translate high-level commands entered by a user (*e.g.*, "PRINT/COPIES = 5 GROCERY.LIS," to print five copies of a grocery list file) into specialized

commands issued to external programs. (*See* Declaration of Douglas Clark, Ph.D ("Clark Decl.") (Ex. 1014) at ¶¶ 25; *see also* Ex. 1007 at 18-19, Section 3.3.1.) Using a "command parse tree" to parse and translate the high-level commands, as recited in the '526 patent, was equally obvious.

This petition relies on prior-art United States Patent No. 6,523,172 ("Martinez-Guerra") (Ex. 1002). Martinez-Guerra is directed to a "parser-translator technology" that translates statements from "a high-level user language" into ones that are "appropriate to a particular data processing application," so that "a user can focus on the semantics of the desired operations and need not be concerned with the proper syntax of a language for a particular system." (*Id.*, Abstract.) As explained below, and in the accompanying Declaration of Dr. Douglas Clark, Martinez-Guerra discloses nearly every limitation of every challenged claim of the '526 patent. And where Martinez-Guerra arguably does not disclose a claim limitation, any necessary modification was well known at the time of the alleged invention and obvious to a person of ordinary skill in the art.

For these reasons, the Board should institute IPR of all the challenged claims, and should find them invalid on the grounds set forth below.

II. MANDATORY NOTICES (37 C.F.R. § 42.8(a)(1))

Pursuant to 37 C.F.R. § 42.8, Petitioner provides the following mandatory disclosures:

A. Real Parties-In-Interest (37 C.F.R. § 42.8(b)(1))

The real party-in-interest is Arista Networks, Inc.

B. Related Matters (37 C.F.R. § 42.8(b)(2))

The '526 patent is at issue in *Cisco Systems, Inc. v. Arista Networks, Inc.*, No. 5:14-cv-05344 (N.D. Cal).

C. Designation of Lead and Back-Up Counsel (37 C.F.R. § 42.8(b)(3))

Petitioner appoints Eugene M. Paige (Reg. No. 55,519) of Keker & Van Nest LLP as lead counsel, and appoints Robert A. Van Nest (*pro hac vice* motion to be filed), Brian L. Ferrall (*pro hac vice* motion to be filed), and David J. Silbert (*pro hac vice* motion to be filed) of Keker & Van Nest LLP as back-up counsel. An appropriate Power of Attorney is filed concurrently herewith.

D. Service Information (37 C.F.R. § 42.8(b)(4))

Service of any documents to lead or back-up counsel may be made to:

Keker & Van Nest LLP
633 Battery Street
San Francisco, CA 94111
(415) 391-5400 (telephone); (415) 397-7188 (fax)

Petitioner consents to service by email at the following addresses: aristaipr@kvn.com; epaige@kvn.com; dsilbert@kvn.com.

III. STANDING (37 C.F.R. § 42.104(a))

Petitioner certifies that (1) the '526 patent is available for *inter partes* review; and (2) Petitioner is not barred or estopped from requesting *inter partes*

review of any claim of the '526 patent on the grounds identified herein. This petition is filed in accordance with 37 C.F.R. § 42.106(a). Concurrently filed herewith is an Exhibit List per 37 C.F.R. § 42.63(e). The Office is authorized to charge the required fee as set forth in 37 C.F.R. § 42.15(a) to Deposit Acct. No. 506260. The Office is further authorized to charge fee deficiencies and credit overpayments to the above-referenced Deposit Account.

IV. OVERVIEW

A. The alleged invention of the '526 patent

The '526 patent describes an alleged improvement for controlling complex administration and/or diagnostic software tools in processor-based systems. (Ex. 1001 at 1:11-14.) It explains that, typically, each of these administration and diagnostic tools has its own command format, which system administrators must remember. (*Id.* at 1:31-34.) Moreover, system administrators may find it difficult "to determine which tool is the best tool (and/or which is the best syntax) to use for a given problem." (*Id.* at 3:16-20.)

To address these problems, the patent proposes a set of "generic commands" that a user can input into a parser-translator system, which will issue a corresponding "prescribed command" to a management program in that program's command format. (*See id.*, Abstract.) As the patent explains, "the new syntax provides a generic instruction set that provides an abstraction of the tool-specific

command formats and syntax, enabling the user to issue command[s] based on the relative functions, as opposed to the specific syntax for a corresponding tool . . . ." (*Id.* at 3:31-35; *see also id.* at 7:1-9:16 (listing "Generic Command Examples").)

The generic command is validated by the system in relation to what the patent calls a "command parse tree." (*Id.* at 1:48-51.) Figure 2, which is reproduced below, depicts the command parse tree (structure 22) of the preferred embodiment:

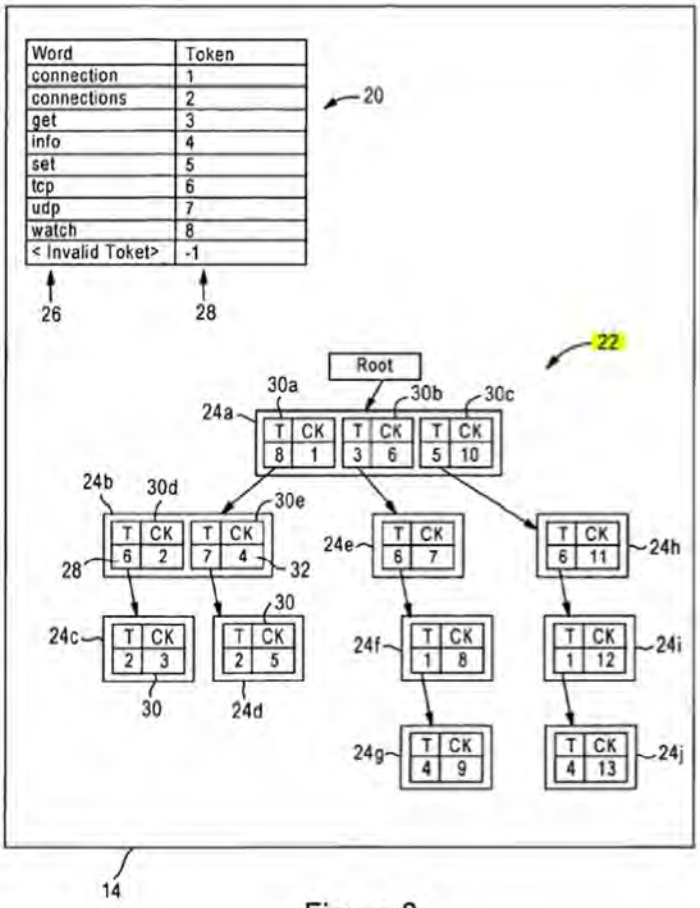


Figure 2

Figure 2 shows a hierarchical data representation that includes multiple elements (structures 24a-j). Each element contains at least one "token"

(abbreviated "T") and a corresponding "command key" (abbreviated "CK"). (*Id.* at 3:51-53.) Figure 2 also depicts a "command word translation table" (structure 20) that matches words within generic commands to tokens, which are shown as numerical values—for example, the generic-command word "watch" corresponds to the token "8." (*Id.* at 3:38-46 and Fig. 2.) These tokens are used to traverse the command parse tree and identify the element that's the "best match" for the received generic command. (*Id.* at 3:47-51.)

Tokens are matched with command-parse-tree elements based on the order of the words in the received generic command. (*Id.* at 4:3-7, 4:14-16.) Beginning at the highest level of the tree, the parser determines if the token corresponding to the first generic-command word is valid. (*Id.* at 4:3-13, Fig. 3.) If so, the parser traverses the next level of elements that depend from the already validated token to determine if the next token in sequence is valid at that level. (*Id.* at 4:13-16, 4:19-27.) Thus, for example, referring to Figure 2, if the first two words in the received generic command are "watch tcp," the parser determines if the token corresponding to "watch" (the number "8") is valid in element 24a, then determines if the token corresponding to "tcp" (the number "6") is valid in element 24b, which depends from token "8" in element 24a.

This process repeats until either no tokens remain, in which case the element corresponding to the final token is identified as the "best match," or the next token

in the sequence is found to be invalid, in which case the element corresponding to the last valid token is identified as the "best match." (*Id.* at 4:27-36, 4:46-49.) The parser then uses the "command key" that corresponds to the "best match" element to identify a "prescribed command" to issue to a selected management program. (*Id.* at 3:51-54, 4:31-36, 4:49-51.) For example, referring to Figure 2, if the parser receives the generic command "watch tcp connections," it will identify element 24c as the "best match," and use the command key "3" to identify a "prescribed command" to issue to a selected management program.

B. The state of the prior art

The idea of translating an input command from a high-level format to a tool-specific format was known for decades before the alleged invention of the '526 patent. (*See* Clark Decl. ¶¶ 20-27.) Since the 1970s, for example, UNIX "shell" programs accepted textual commands from a user and converted them into instructions for an operating system. (*See* Dennis M. Ritchie & Ken Thompson, The UNIX Time-Sharing System, 17 Communications of the ACM 365 (Jul. 1974) (Ex. 1012) at 7.) The UNIX shell and other programs converted user-input commands using a parser, which compared a statement to the rules of a language (known as its "grammar") and interpreted the statements. (*See* Brian W. Kernighan & Rob Pike, The UNIX Programming Environment (1984) (Ex. 1009) at 87.) Parsers are elemental to computer science and were well-known long before the

'526 patent was applied for. (*See* Alfred V. Aho, et. al., Compilers: Principles, Techniques, And Tools (1986) (Ex 1005) at 52-60. To cite just one example, in a 1974 paper, Ritchie and Thompson explained that the UNIX shell parses commands and accompanying "arguments" (*i.e.*, parameters) that a user types into the command line, noting that "[t]he Shell is a command line interpreter: it reads lines typed by the user and interprets them as requests to execute other programs." (Ex. 1012 at 7.)

The UNIX shell and related systems received commands input by a user (including commands in a high-level command format), parsed the commands, and issued appropriate procedure calls to external binary files. *Id.* Many other systems did the same thing. As noted, for example, Digital Equipment Corporation's "Digital Command Language," or "DCL," was used since the 1980s to translate high-level commands entered by a user (*e.g.*, "PRINT/COPIES = 5 GROCERY.LIS," to print five copies of a grocery list file) into specialized commands issued to external programs. (Ex. 1007 at 18-19, Section 3.3.1; *see also* Clark Decl. at ¶ 25.)

The use of a "command parse tree" to parse sequences of tokens, as recited in the'526 patent, was also obvious to persons of skill in the art by the time the patent was applied for. Noam Chomsky's foundational work on formal languages in the 1950's proposed a tree representation:

> Every [finite-state, or regular] language can be represented in the form of a *tree*. At the root of the tree is a point from which all sentences start. Each word that can occur as the initial word in some sentence is represented by a branch leaving this initial point. At the end of the branch representing any particular word will be another set of branches representing each of the words that can follow the first. This process of arborization continues until every possible sentence is represented by some path through the tree; or, if the language contains an infinite set of sentences, the tree will continue indefinitely.

(Ex. 1003 at 4.) Skilled artisans therefore understood that any command language that has a finite number of possible commands can be represented as a tree.

Parsers of such languages process commands left-to-right, at each step consulting a data structure that represents the language's grammar, to see what command words are acceptable next choices. For finite command languages, these data structures are hierarchical representations of the language ("command parse trees"), as the '526 patent discloses. (Clark Decl. at ¶ 22-23.) As explained, for example, in U.S. Patent No. 6,134,709, applied for in June 1998, in its "Background of the Invention" section, "[t]he list of acceptable commands can be stored in a tree structure to reduce the space required to store all of the acceptable commands. The root nodes of the tree can contain acceptable first tokens in a command. . . . The tree can be searched to match the submitted command, token by token." (U.S. Patent No. 6,134,709 to Pratt, issued Oct. 17, 2000 (Ex. 1013) at 1:40-57; Clark Decl. at ¶ 23.)

This petition explains why the claims of the '526 patent are unpatentable over the prior-art Martinez-Guerra patent, applied for in February 1999, which is described in more detail in Section VIII below.

V. LEVEL OF ORDINARY SKILL IN THE ART

A person of ordinary skill in the art ("POSA") is presumed to be aware of all pertinent art, follows the conventional wisdom in the art, and is a person of ordinary creativity. With respect to the '526 patent, a POSA would typically have at least a Bachelor's Degree in computer science and 3-5 years of experience in systems development. (Clark Decl. ¶¶ 15-16.)

VI. CLAIM CONSTRUCTION

The Board gives claims their broadest reasonable interpretation ("BRI") consistent with the specification. 37 C.F.R. § 42.100(b); *In re Cuozzo Speed Technologies, LLC*, 793 F.3d 1268, 1279 (Fed. Cir. 2015).[1] Such a construction must reasonably reflect the plain language and disclosure of the patent "as [they]

[1] Other forums, such as district courts, apply a different standard. *See Phillips v. AWH Corp.*, 415 F.3d 1303 (Fed. Cir. 2005) (*en banc*). Any interpretation of the challenged claims in this petition, either explicit or implicit, does not reflect Arista's interpretation under a standard other than the BRI. In particular, Arista reserves the right to argue for different constructions under the *Phillips* standard in the related district-court litigation.

would be interpreted by one of ordinary skill in the art." *In re Suitco Surface Inc.*, 603 F.3d 1255, 1259 (Fed. Cir. 2010).

As relevant here, the Board should construe the claim terms as follows:

A. "management program"

Proposed construction: "separate tools or external agents having their own respective command formats that provide management functions."

Cisco Systems, Inc. ("Cisco"), the owner of the '526 patent, has proposed the above construction for "management program" in the related litigation. (Cisco Preliminary Claim Construction Disclosure, *Cisco Systems, Inc. v. Arista Networks, Inc.*, No. 5:14-cv-05344-BLF (N.D. Cal. Aug. 24, 2015) (Ex. 1010) at 3.) For the purpose of this proceeding, Arista agrees that Cisco's construction would be the broadest reasonable interpretation of this term. The '526 patent states that management programs "may be executed within the processor based system or externally as external agents," and describes them as having their own "respective command formats and syntax." (Ex. 1001 at 3:1-5, 1:41-44; *see also* Clark Decl. ¶ 29.)

B. "command parse tree"

Proposed construction: "a hierarchical data representation having elements each specifying at least one corresponding generic command component and a corresponding at least one action value."

Cisco has proposed the above construction for "command parse tree" in the related litigation. (Ex. 1010 at 10.) For the purpose of this proceeding, Arista agrees that Cisco's construction would be the broadest reasonable interpretation of this term. Although a "parse tree" commonly refers to the result of parsing an input stream into its constituent parts, a skilled artisan would understand that the broadest reasonable interpretation of the phrase "command parse tree" in the '526 patent is not the result of a parsing operation; it is instead a tree representation of the entire language, used to perform a parsing operation, as illustrated in Figure 2 of the patent. (*See* Ex. 1001 at 3:39 (referring to "command parse tree 22"); *see also* Clark Decl. ¶ 30.)

C. "recursively traversing"

Proposed construction: "traversing using a process that repeats itself"

The '526 patent uses the term "recursively traversing" only once outside the claims, stating that "parser 14 recursively traverses the command parse tree 22 for each command word to identify the best match for the generic command." (Ex. 1001 at 3:55-57.) Figure 3 and the descriptions thereof explain the process of "traversing" the command parse tree: the parser repeats a series of steps in a looped manner for each successive command word. (*See, e.g.*, *id.* at 4:10-11, 4:19-23, Fig. 3.) Because Figure 3 shows a process by which a set of steps is repeated in a looped manner, one of skill in the art would understand that the broadest

reasonable interpretation of "recursively traversing" includes such a looping mechanism. (Clark Decl. ¶¶ 31-32.)

D. "means for validating" / "validating means"

This claim phrase is subject to 35 U.S.C. § 112(f) (formerly 35 U.S.C. § 112, ¶ 6).

Function: "validating a generic command received from a user."

Corresponding Structure: Parser 14 of Figures 1 and 2, as described in 3:36-61, and executing an algorithm as disclosed in Figure 3.

The '526 patent states that "[t]he parser 14 is configured for validating a received generic command by comparing each input command word to the command parse tree 22 to determine for the received generic command a tree element 24 identified as a best match." (Ex. 1001 at 3:47-51; *see* also Clark Decl. ¶¶ 33-34.)

VII. SUMMARY OF THE PRIOR ART FORMING THE BASIS OF THIS PETITION

U.S. Patent No. 6,523,172 to Martinez-Guerra *et al.*, entitled "Parser Translator System and Method" ("Martinez-Guerra") (Ex. 1002), was filed February 19, 1999, and issued February 18, 2003. Martinez-Guerra is thus prior art to the '526 patent under 35 U.S.C. § 102(e).

Martinez-Guerra, which was not before the Office during prosecution of the '526 patent, is directed to a "parser-translator technology" that allows users to

enter statements "in a high-level language," and translates those statements into "directives appropriate to a particular data processing application." (Ex. 1002, Abstract; *see* also Clark Decl. ¶ 36.) As Martinez-Guerra explains, "[u]sing the parser-translator technology, a user can focus on the semantics of the desired operations and need not be concerned with the proper syntax of a language for a particular system." (Ex. 1002, Abstract.)

Martinez-Guerra teaches that its parser-translator can be used to control multiple programs, each with its own grammar. (*Id.*) For example, it teaches that "a single parser-translator component provides multiple software applications with an interface to high-level user language statements wherein operation of the single parser-translator component is suitably defined for each software application using a corresponding grammar encoding." (*Id.* at 3:48-53; *see also, e.g., id.* at 13:34-42 (describing how "tool-specific translation rules" enable "a parser-translator component implemented in accordance with the present invention [to] be used to provide a natural language dialogue facility in a wide variety of software tool environments."); Clark Decl. ¶ 36-37.)

With respect to the parser itself, Martinez-Guerra teaches that "[a] variety of token recognizer . . ., parser . . ., and translator . . . designs are suitable," and describes exemplary embodiments. (Ex. 1002 at 10:31-41.) The embodiments employ a grammar that includes "phrase structure rules," which "describe the legal

ordering of tokens in a language"; "dictionary entries," which "describe all the tokens that a parser-translator . . . should recognize"; and "mapping rules," which "indicate how a sequence of tokens should be translated by the parser-translator[.]" (*Id.* at 14:8-21; *see* also Clark Decl. ¶ 38.) The "mapping rules" are also referred to as "translation rules." (Ex. 1002 at 15:23.)

As each word in a command is parsed, these rules define a set of valid next tokens in light of the tokens that have already been validated. (*See, e.g., id.* at 18:36-40 ("Token recognizer 31 computes a set of valid next states . . . [that] includes the set of next tokens consistent with a current parse state."); *see* also Clark Decl. ¶ 45.) At first, the set of valid tokens includes all "tokens that may begin a legal statement in the language defined by the grammar." (Ex. 1002 at 18:40-42.) Each successive token is then compared to the set of valid next tokens in light of the tokens that came before. (*See, e.g., id.* at 20:31-34 ("Token recognizer 31 performs input matching against each of the valid next states corresponding to a parse state representation consistent with the input stream read so far."); *see* also Clark Decl. ¶ 45.) Finally, the complete command is translated and the translated command is issued to the appropriate program. (*See, e.g.,* Ex. 1002 at 20:61-21:10 and Fig. 1.)

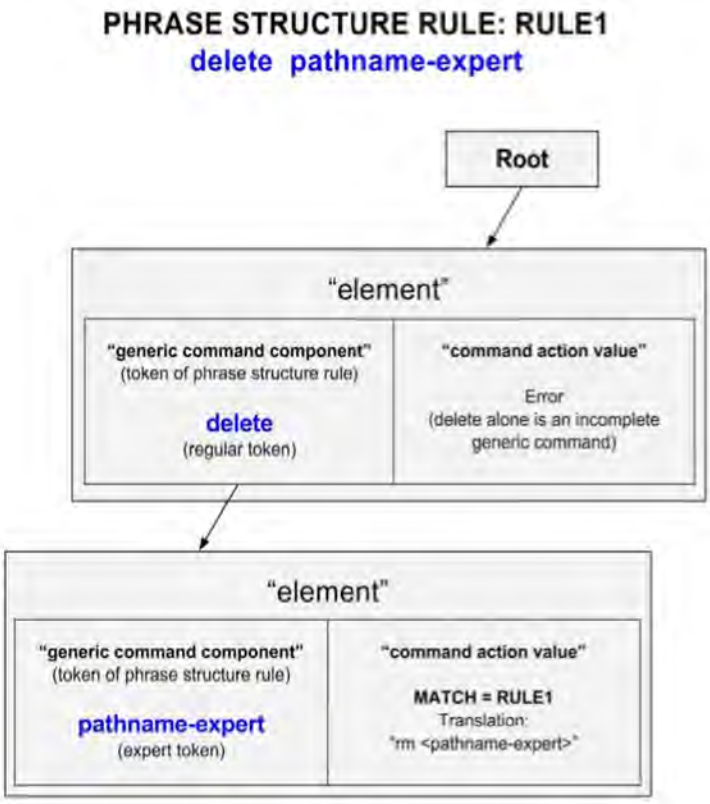
As an illustrative example, Martinez-Guerra explains how a parser-translator would translate the command "delete /usr/extract/testing" received from a user,

where "/usr/extract/testing" is the name of a file. (*See* Ex. 1002 at 15:50-16:5.) The parser-translator uses the hypothetical phrase structure rule "RULE1," which describes the legal ordering of tokens: "delete pathname-expert." (*Id.*; *see* also Clark Decl. ¶ 41.) "[D]elete is a regular token matching the string delete in the input stream[.]" (Ex. 1002 at 15:58-59.) "/usr/extract/testing" is what the patent calls an "expert token," which is "a string whose value cannot be specified when the grammar is written but which can be defined in terms of its characteristics." (*Id.* at 15:4-6, 15:67-16:5.) Thus, "pathname-expert" corresponds to the pathname of a file. (Clark Decl. ¶ 42.)

When the command "delete /usr/extract/testing" is received, each word is matched to a corresponding token found in the dictionary entries. (Ex. 1002 at 9:34-38, 14:60-15:15; Clark Decl. ¶ 42.) Martinez-Guerra explains that the dictionary entries identify all tokens that the parser-translator should recognize; thus, they act as a table for correlating an input word with a corresponding token. (*Id.* at 14:16-18, 14:60-62, 20:22-24; Clark Decl. ¶ 38, 41.) The tokens are then compared to the phrase structure rules. (*Id.* at 11:51-60; Clark Decl. ¶ 40-41.) Because "/usr/extract/testing" is recognized as a file pathname, the parser-translator will look for "pathname-expert" in place of "/usr/extract/testing." (*Id.* at 15:54-16:5; Clark Decl. ¶ 42.)

If the sequence of tokens matches a sequence in a phrase structure rule, then that rule is satisfied, and the parser-translator issues the corresponding translation. (Ex. 1002 at 9:40-43, 20:45-48; Clark Decl. ¶ 43.) In the example above, because the tokens "delete" and "pathname-expert" satisfy RULE1, the translator (if in a UNIX environment) translates the command as "rm <pathname-expert>," where "<pathname-expert>" will be replaced with the original user input. (Ex. 1002 at 15:59-16:5; Clark Decl. ¶ 43.) Thus, the parser-translator issues the prescribed command "rm /usr/extract/testing," directing the system to remove (*i.e.*, delete) the specified file. (Ex. 1002 at 16:4-5; *see also* Clark Decl. ¶ 43, 61.)

For any finite command language as described in the '526 patent, which are a subset of the languages addressed by Martinez-Guerra, Martinez-Guerra's phrase structure rules constitute a command parse tree because they are collectively a hierarchical data representation of the valid sequences of components of statements in a high-level user language, and because every component of each phrase structure rule has an action that will occur if that component is the last valid component parsed. For example, the phrase structure rule RULE1 would be represented as a set of component-action value pairs (what the '526 patent calls "elements") within such a command parse tree as illustrated in the figure below:



(Clark Decl. ¶¶ 38-39.)

VIII. IDENTIFICATION OF CHALLENGE (37 C.F.R. § 42.104(b))

Petitioner requests *inter partes* review and cancellation of claims 1-26 on the grounds that they are obvious over Martinez-Guerra. As set forth below, Petitioner believes that Martinez-Guerra discloses every limitation of claims 1-6, 8, 10-19, 21, and 23-26. However, because some disclosures may be seen as requiring corroboration from the expert, Petitioner challenges these claims under obviousness rather than anticipation. Claims 7, 9, 20, and 22 recite a single limitation that Martinez-Guerra does not expressly disclose, but that was

unquestionably obvious: that the management program *executes* the prescribed command it receives.

In support of the grounds for unpatentability, this petition is accompanied by the Declaration of Dr. Douglas Clark (Ex. 1014), who explains what Martinez-Guerra would have conveyed to a POSA, and why Martinez-Guerra discloses or renders obvious every challenged claim element.

A. Claim 1 and dependent claims 2-9 are invalid as obvious over Martinez-Guerra

1. [1A] "A method in a processor-based system configured for executing a plurality of management programs according to respective command formats, the method comprising"

Martinez-Guerra discloses a parser-translator that enables a high-level user language to interface with multiple software applications. (Ex. 1002 at 3:48-53; Clark Decl. ¶ 47.) It does so by translating statements that a user enters in a high-level language (which can include generic commands) to "logically and syntactically correct directives for performing the desired data transformations or operations" (prescribed commands) for the proper software tool. (Ex. 1002 at 3:29-37; Clark Decl. ¶ 47.) The method is carried out in software executed on a computer system. (Ex. 1002 at 9:10-13; Clark Decl. ¶ 47.)

The broad disclosure of software tools that can be used with the system includes "management programs." For example, Martinez-Guerra explains that "an illustrative enterprise tools environment includ[es] data discovery and cleansing

tools, data extraction conversion and migration tools, data movement and replication tools, query Multi-Dimensional Data (MDD) analysis and On-Line Analytical Processing (OLAP) tools, as well as applications and gateways that may advantageously incorporate a parser-translator component." (Ex. 1002 at 11:36-47; *see also id.* at 13:14-20 (additional software applications, including "query and on-line analytical processing tools, gateways to operational data systems"); Clark Decl. ¶ 48.)

The disclosed tools are separate from the parser-translator and from one another, and as illustrated by Martinez-Guerra's example of translating a "delete" command, use a command format that may differ from the high-level user language command provided by the user in the input stream. (*See* Ex. 1002 at 15:50-62 (translating the high-level input command "delete /usr/extract/testing" to the prescribed UNIX command "rm /usr/extract/testing"); *see also, e.g., id.* at 3:48-53 (describing a "corresponding grammar encoding" for "each software application"), 13:34-42 (describing "tool-specific translation rules," indicating that the tools have respective command formats); Clark Decl. ¶ 49.)

2. [1B] "receiving a generic command from the user;"

Martinez-Guerra discloses a user interface that receives from a human user an input stream consisting of statements in a "high-level user language[.]" (Ex. 1002, Abstract; *see also id.* (users can focus on "the semantics of the desired

operations" when inputting commands), Fig. 3 (item 36) (input stream), 9:15-23 (input "supplied by a human user"); Clark Decl. ¶ 50.) One of ordinary skill in the art would understand that such statements, as described in Martinez-Guerra, can be "generic commands" within the meaning of the '526 patent, because they are abstractions of desired operations that are later translated into directives for specific software tools. (Clark Decl. ¶ 51; *see also* Ex. 1002 at 10:7-11, 13:30-42.) For example, one of ordinary skill in the art would understand that the command "delete /usr/extract/testing" in Martinez-Guerra is analogous to the "Generic Command Examples" disclosed in the '526 patent. (*See* Ex. 1002 at 15:50-62; Ex. 1001 at 7:1-9:15 (identifying, *e.g*, "set watchtime <milliseconds>" as an exemplary generic command).)

3. [1C.1] "validating the generic command based on a command parse tree that specifies valid generic commands relative to a prescribed generic command format,"

Martinez-Guerra discloses the use of a command parse tree as claimed in the '526 patent. Martinez-Guerra explains that words in the input stream are analyzed by the token recognizer subcomponent of the parser-translator according to rules encoded in the grammar. (*See* Ex. 1002 at 10:42-58; Clark Decl. ¶ 52.) Martinez-Guerra defines "grammar" as "a formal declaration of the syntactic structure of a language typically represented as a set of rules that specify legal orderings of constituents and subconstituents (*e.g.*, phrases and symbols) in a language."

(Ex. 1002 at 7:52-56.) The "legal orderings" refer to valid sequences of tokens as defined by phrase structure rules in the grammar. (*Id.* at 8:24-26.) The grammar also provides translation functions corresponding to all such valid sequences of tokens. (*Id.* at 16:15-16, 17:51-52; Clark Decl. ¶ 52.)

Martinez-Guerra explains that the parser-translator creates "internal data structures," including "parse state data structures" and "internal representations of a grammar," which it uses to specify "next legal choices" allowed by the grammar. (*Id.* at 18:31-40 and 10:45-46; Clark Decl. ¶ 53.) "[T]he set of valid next states includes the set of next tokens consistent with the current parse state." (Ex. 1002 at 18:38-40.) As each token is ingested, the parser-translator uses the phrase structure rules encoded in the grammar to specify the possible valid next legal choices until the input stream has been fully parsed. (Clark Decl. ¶ 53; Ex. 1002 at 10:42-58, 20:45-49.) For any finite high-level command language, these phrase structure rules encoded in the grammar thus constitute a command parse tree, because they are collectively a hierarchical data representation of the valid components of statements in a high-level language, and as explained below, their elements specify at least one corresponding generic command component and a corresponding action value. (Clark Decl. ¶¶ 21-24, 53.)

When the input stream contains a generic command, the command is validated by the parser-translator relative to a prescribed generic command format

defined by the grammar. (*Id.* at 10:42-58; Clark Decl. ¶ 54.) Specifically, the token recognizer subcomponent receives the initial command word (*e.g.*, "delete") and first determines whether the word matches a token in the dictionary entries, which define all tokens found in the phrase structure rules of the grammar. (Clark Decl. ¶ 54; Ex. 1002 at 9:34-38, 14:16-18.)

Once it has confirmed a match, the token recognizer then determines whether the token corresponding to the command word is present among the next legal parse states maintained by the parser. (Clark Decl. ¶ 54; Ex. 1002 at 9:34-40, 20:28-34.) As discussed above, each parse state identifies valid next legal tokens, as defined by the phrase structure rules. (Clark Decl. ¶ 55.) In the example of column 15:50-62, the tokens "delete" and "pathname-expert" are matched, respectively, with the first and second legal parse words of the phrase structure rule, "RULE1." (Clark Decl. ¶ 55; Ex. 1002 at 15:50-62 ("RULE1 → delete pathname-expert").) Because the tokens "delete" and "pathname-expert," in that order, match the phrase structure rule RULE1, the generic command is validated. (Clark Decl. ¶ 55.)

4. [1C.2] "the command parse tree having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value,"

As discussed above, Martinez-Guerra explains that the grammar includes "a set of phrase structure rules and associated translation rules that define the

syntactic order of a source and target language . . . ." (Ex. 1002 at 7:57-59; *see also* 16:15-16.) And for any finite command language, the phrase structure rules encoded in the grammar constitute a command parse tree, because they are collectively a hierarchical representation of the valid components of statements in a high-level language, and as discussed below, their elements specify at least one corresponding generic command component and a corresponding action value. (Clark Decl. ¶¶ 21-24.) Phrase structure rules define a legal ordering of tokens in a particular language (*id.* at 14:11-12), and can be represented as textual rule specifications (*e.g.*, "RULE1 → delete pathname-expert"). (Clark Decl. ¶ 56.) In the textual rule specification for RULE1, each token to the right of the arrow is a component needed to form a complete statement; each of them is what the '526 patent calls a "generic command component." (*Id.*)

As claimed in the '526 patent, Martinez-Guerra discloses that each component of the phrase structure rule has a corresponding appropriate action that will occur if that component is a "best match" for the generic command. If an input word is invalid or a command is incomplete—for example, if only the word "delete" is received without any pathname to define the target of the operation—that action may result in returning an error. (*See* Ex. 1002 at 19:3-4.) On the other hand, a valid input stream that matches and completes a phrase structure rule (*e.g.*, "delete /usr/extract/testing" (*id.* at 16:5)) will correspond to an action of the